第34卷 第17期 计 算 机 工 程 2008年9月
Vol.34 No.17 Computer Engineering September 2008
·软件技术与数据库· 文章编号：1000—3428(2008)17—0034—03 文献标识码：A 中图分类号：TP311

嵌入式空间数据库综合查询算法

刘 平，陈旭灿，李思昆
(国防科技大学计算机学院，长沙 410073)

摘 要：嵌入式空间数据库一般作为嵌入式GIS的后端，为其提供对空间数据和属性数据的存储、搜索、查询等多项功能。其中，查询性能是直接影响嵌入式GIS运行效率的基本因素之一。该文对嵌入式空间数据库综合查询算法进行分类，提出并实现了先空间串行查询算法、先属性串行查询算法和并行查询算法，对该3种查询算法进行性能测试与比较，并给出了测试比较结果。

关键词：嵌入式空间数据库；综合查询算法；嵌入式GIS

Integrated Query Algorithm of Embedded Spatial Database

LIU Ping, CHEN Xu-can, LI Si-kun
(School of Computer Science, National University of Defense Technology, Changsha 410073)

【Abstract】 As an embedded GIS backend, embedded spatial database provides the spatial data and attribute data storage, search, query and other functions. Among them, query performance is one of the basic direct factors impact on the operating efficiency of embedded GIS. This paper talks about the classification of the embedded spatial database integrated query algorithms, puts forward and realizes the spatial first serial query algorithm, the attribute first serial query algorithms and the parallel query algorithm. These three query algorithms are tested and compared, and the test results are given.

【Key words】 embedded spatial database; integrated query algorithm; embedded GIS

1 概述

嵌入式空间数据库是指在嵌入式系统中，能够存储由空间地理信息和属性数据组成的空间数据，并至少提供高效的空间索引和空间操作的数据库管理系统[1]。由于嵌入式资源非常有限，嵌入式空间数据库一般作为嵌入式GIS的后端，为嵌入式GIS提供对空间数据和属性数据的存储、搜索、查询等多项功能[2]。其中，查询是嵌入式GIS利用嵌入式空间数据库所进行的最基本且最频繁的操作之一，在某种程度上，其性能也直接影响了嵌入式GIS的运行效率。

目前，在空间数据库的查询算法研究上，都把提高查询响应时间考虑在空间数据或属性查询单一的查询方式上，并且涌现出了众多的优秀查询算法。例如，空间查询算法有：基于R树，R+树，R*树，K-D树，GiST索引[3]等多种索引查询算法；属性查询算法有：基于B树，B+树[4]等多种索引查询算法。且这些算法还在进一步的探索和研究中。而对空间数据和属性数据的综合查询算法研究甚少，即使有些系统能够完成属性和空间数据的综合查询，大多也是基于文件系统的空间数据，由嵌入式GIS自身所集成的功能完成，对查询算法并没有形成成熟的算法理论。对于处在发展初期的嵌入式空间数据库来说，更是停留在理论发展的雏型阶段。而在嵌入式空间数据库中，空间数据是多维数据，属性数据是一维数据，系统不能采用一致的查询方式对它们进行处理。因此，研究具体按什么样的顺序对空间数据和属性数据进行综合查询的算法对于提高系统的性能具有很重要的意义。

2 综合查询算法分类

嵌入式空间数据库综合查询算法既包括对空间数据的空间查询又包括对属性数据的属性查询。按其对属性数据和空间数据的处理顺序的不同，可将其大体分为2类：串行查询和并行查询。

按其算法的系统实现方式又可细分为3种：

(1)先空间查询，系统先进行空间运算，然后根据空间运算返回的结果和属性约束条件进行属性查询。

(2)先属性查询，系统首先根据属性约束条件进行的属性查询，然后根据属性查询返回的结果和空间约束条件进行空间查询。

(3)空间和属性同时查询，系统首先根据属性和空间约束条件同时对属性和空间进行查询，然后对查询结果求交集，以获得最终查询结果。

3 先空间串行查询算法

3.1 基本思路

用户首先通过应用程序用户界面选择一个查询区域，输入属性约束条件，接着应用程序内部通过与嵌入式空间数据库的接口，利用嵌入式空间数据库调用嵌入式空间数据库内部空间查询模块的空间操作函数和空间索引模块的R*树空间索引进行空间查询，然后将空间查询结果集返回给属性查询部分，再接着属性查询函数在这些对象上，根据用户输入的属性约束条件，调用底层的嵌入式关系数据库SQLite内部机制B+树索引，进行属性查询。最后将查询结果集以表格或可视化的形式呈现给用户。

3.2 算法形式化描述

设$\exists t \in r[Q(t)]$表示集合r中存在元素t使得公式Q为真，则用$\{t \mid \exists t \in r[Q(t)]\}$表示关系中满足查询条件$Q(t)$的元组集

基金项目：国家自然科学基金资助项目(90207019)
作者简介：刘 平(1983－)，女，硕士研究生，主研方向：计算机体系结构；陈旭灿，副研究员；李思昆，教授、博士生导师
收稿日期：2007-09-20 **E-mail：**lp1142003@yahoo.com.cn

合。其中，r 表示嵌入式空间数据库中的关系表；t 表示关系表中的元组；Q 表示界面程序输入的查询条件。

下面给出基于双表的先空间查询算法的形式化定义。其中，$r\infty s$ 为关系表 r 和 s 的连接；$(r\infty s)'_{spatial}$ 为进行空间查询所得出的元组集合；$Q(t)$ 为综合查询条件；$Q_{spatial}(t)$ 为空间查询条件；$Q_{attribute}(t)$ 为属性查询条件，则先空间查询算法可分步表示为

(1) $(r\infty s)'_{spatial}=\{t\mid\exists t\in(r\infty s)[Q_{spatial}(t)]\}$

(2) $\{t\mid\exists t\in(r\infty s)[Q(t)]\}=\{t\mid\exists t\in(r\infty s)'_{spatial}[Q_{attribute}(t)]\}$

根据上述算法的基本思路，形式化定义其算法实现流程如下：

```
Begin:
Input 查询范围 Region, 属性约束条件 attConstraint, 查询类型 Query-Type, 访问指针 Vistor;
For  元组 r  in  关系 t_r  do  begin
 SpatialQuery(Region,Query-Type, Vistor){
   Switch(Query-Type){
    “intersect”:intersectsWithQuery(Region,Vistor);
    “10NN”: nearestNeigbourQuery(Region, Vistor);
    “selfjion”: selfJoinQuery(Region, Vistor);
    …}
   }
End;
If (Vistor 所指向的结果集为空)
  Return  0;
Else
  For  元组 r  in  关系 t_r  do  begin
    attributeQuery( Vistor 返回的结果集, attConstraint){
      Sqlite3_get_table (数据库名，由属性条件和返回的结果集组成的查询语句，&结果集，返回结果的行，返回结果的列，错误报告);
        Return 最终结果集;}
  End;
End.
```

该方法比较符合用户的思维习惯，实现起来也方便。但是由于空间查询需要操作大量复杂的空间对象，同时需要用到计算量极大的算法，因此它既是 CPU 密集型又是 I/O 密集型的，而属性查询只需操作大量简单的一维数据，仅属于 I/O 密集型的。且通常在嵌入式空间数据库中对空间查询函数处理的 CPU 代价远远高于 I/O 代价，于是将空间查询放在属性查询的前面，特别是对于存储资源和计算资源有限的嵌入式 GIS 应用来说，势必影响查询操作的效率。

4 先属性串行查询算法

4.1 基本思路

用户首先通过应用程序用户界面选择一个显示区域，输入属性约束条件，接着应用程序内部通过与嵌入式空间数据库的接口，利用嵌入式空间数据库调用嵌入式空间数据库内部空间查询模块空间过滤器中的空间过滤操作函数，过滤掉空间查询约束条件，再接着利用底层嵌入式关系数据库 SQLite 的内部机制 B+树索引，对过滤后的数据进行属性查询。然后嵌入式空间数据库内部空间查询模块的空间运算函数根据属性查询返回的结果集和过滤掉的空间查询约束条件，利用 R*树空间索引进行空间查询，返回满足条件的空间对象集，最后将查询结果集以表格或可视化的形式呈现给用户。

4.2 算法形式化描述

设 $r\infty s$ 为关系表 r 和 s 的连接，$(r\infty s)'_{attribute}$ 为进行属性查询所得出的元组集合，$Q(t)$ 为综合查询条件，$Q_{spatial}(t)$ 为空间查询条件，$Q_{attribute}(t)$ 为属性查询条件，则其算法可以分步表示为

(1) $(r\infty s)'_{attribute}=\{t\mid\exists t\in(r\infty s)[Q_{attribute}(t)]\}$

(2) $\{t\mid\exists t\in(r\infty s)[Q(t)]\}=\{t\mid\exists t\in(r\infty s)'_{attribute}[Q_{spatial}(t)]\}$

根据上述所描述算法的基本思路，形式化定义其算法实现流程如下：

```
Begin:
input 查询范围 Region, 属性约束条件 attConstraint, 访问指针 Vistor, 查询类型 Query-Type;
For  元组 r  in  关系 t_r  do  begin
  spatialFliterFuntion(Region,attConstraint);
    attributeQuery(spatialFileterFuntion 返回的只有属性约束条件的查询语句){
      Sqlite3_get_table(数据库名，由属性条件组成查询语句，&返回的结果集，返回结果的行，返回结果的列，错误报告);}
End;
If (返回的结果集为空)  Return  0 ;
Else
  For  元组 r  in  关系 t_r  do begin
    SpatialQuery(过滤掉的查询条件，Query-Type，Vistor，属性查询返回的结果集){
      Switch(Query-Type){
        “intersect” : intersectsWithQuery(Region,Vistor);
        “10NN”:  nearestNeigbourQuery(Region, Vistor);
        “selfjion”: selfJoinQuery(Region, Vistor);
        …}
  End;
End.
```

该方法同先空间串行查询算法相比，在实现上增加了一个空间过滤器(用于过滤掉输入条件中的空间约束条件)，因此，在某种程度上，会降低系统的处理效率。不过，由于属性查询是 I/O 密集型的，而空间查询既是 CPU 密集型的又是 I/O 密集型的，因此将属性查询算法放在前面执行，可以使得在进行空间查询之前过滤掉一些计算量极大的运算，同时弥补增加空间过滤器和系统计算资源不足所带来的查询操作的效率问题，但是增加了 I/O 资源的负担。

5 并行查询算法

由于本查询算法是基于 Linux 进行研究和实现的，因此可以利用 Linux 环境下的 POSIX 多线程编程机制，使空间查询和属性查询能够进行并行操作，以提高系统内部的查询响应时间。

5.1 POSIX 多线程

POSIX 多线程是提高代码响应和性能的有力手段。主要特征表现如下：

(1)操作系统内核开销小。线程相对于进程而言是一种非常“节俭”的多任务操作方式。同一进程中的不同线程可以使用相同的地址空间，共享大部分的数据，线程的启动开销和线程间的切换开销也远远小于进程。

(2)程序复杂性小。不用编写 IPC 代码。

(3)快捷灵活性。线程启动开销很小，切换迅速，使得可以大量使用线程而无须太过于担心带来 CPU 或内存不足。这点特别适合嵌入式研究。

(4)可移植性。顾名思义，POSIX 即可移植操作系统接口，用它编写的代码可运行于 solaris, freebsd, Linux 等其他多种平台。

除了上述特征外，POSIX 多线程还采用了线程数据、互斥锁、条件变量和信号量的机制来达到多线程的目的。

5.2 并行查询算法实现

5.2.1 基本思路

用户首先从应用程序界面选择显示范围，输入属性约束条件，然后应用程序内部采用 POSIX 多线程机制，通过与嵌入式空间数据库的接口，同时调用嵌入式空间数据库内部空间查询模块的空间查询函数和空间索引模块的 R*树空间索引和底层嵌入式数据库 SQLite 的 B+树索引进行空间和属性的双向查询，接着对它们返回的结果集求交集，最后将求交所得的最终查询结果集以表格或可视化的形式返回到用户界面。

5.2.2 算法形式化描述

设 $r\infty s$ 为关系表 r 和 s 的连接，$Q(t)$ 为综合查询条件，$Q_{spatial}(t)$ 为空间查询条件，$Q_{attribute}(t)$ 为属性查询条件，而 $Q_{spatial}(t)\bigcap Q_{attribute}(t)=\varnothing$，则

$$\{t\mid\exists t\in(r\infty s)[Q(t)]\}=\{t\mid\exists t\in(r\infty s)[Q_{spatial}(t)]\}\bigcap\{t\mid\exists t\in(r\infty s)[Q_{attribute}(t)]\}$$

根据上述所描述的算法的基本思路，形式化定义其算法实现流程如下(实现中需包含头文件名 pthread.h)：

```
#include <pthread.h>
Begin:
声明 2 个线程句柄 pthead_t  spatial, attribute;
input 查询范围 Region, 属性约束条件 attConstraint, 查询类型 Query-Type, 访问指针 Vistor ;
For 元组 r in 关系 t_r do begin
  Pthead_create(&spatial,NULL,(void)SpatialQuery,NULL);
  Pthread_create(&attribute,NULL,(void)attributeQuery,NULL);
  Return spatialQuery.结果集与 attributeQuery.结果集的交集;
End;
SpatialQuery (Region, Query-Type, Vistor){
  Switch(Query-Type){
    “intersect”:  intersectsWithQuery(Region, Vistor);
    “10NN”:  nearestNeigbourQuery(Region, Vistor) ;
    “selfjion”:  selfJoinQuery(Region, Vistor);
    …}
  Return 结果集; }
attributeQuery( Vistor 返回的结果集, attConstraint){
  Sqlite3_get_table(数据库名, 由属性条件和返回的结果集组成的 sql 语句, &结果集, 返回结果的行, 返回结果的列, 错误报告);
  Return 结果集; }
End.
```

该方法采用了 POSIX 多线程编程机制，使得空间查询和属性查询操作可以同时进行。较前 2 种算法而言，理论上具有更高查询效率和更好的移植性。但是由于空间查询和属性查询的同时进行，需要处理的记录数也较前 2 种查询算法多。当记录数超过一定数量而空间查询函数的 CPU 代价较低时，可能会影响系统的查询效率。

6 性能测试与比较

衡量一种查询算法的优劣，可以通过该查询算法执行时对系统各种资源的使用情况来考虑。由于嵌入式系统本身的特殊性，因此从嵌入式软件和查询算法本身这 2 方面入手。一方面，作为嵌入式软件，由于嵌入式系统本身的设备资源非常有限，要求它占有更少的内存空间，消耗更低的 CPU 资源，具有很高的可靠性；另一方面，作为查询算法本身，则要求它支持更大容量的数据库、更快的查询响应时间和更强的事物处理能力。然而，对这 2 方面的要求，在某些情况下又是冲突的。比如，操作大容量的数据库显然需要更多的内存和 CPU 资源。为了更好地测量各种算法的查询效率，综合上述因素，从平均查询响应时间的角度，对上述 3 种综合查询算法进行测试与分析比较。

6.1 测试数据

测试数据取自 17 套中国沿海海图数据，海图以图幅-图层形式组织。海图数据格式与 shapefile 格式基本相同，且其空间数据和属性数据均存放在嵌入式空间数据库中。

6.2 测试环境

硬件运行环境：以 ARM920T 为嵌入式处理器为核心的三星 SMDK 开发板。软件运行环境：MontaVista Linux 专业版(Professional Edition)嵌入式 Linux 实时操作系统，基于 QTE 图形界面开发包开发的嵌入式态势标绘软件系统 QGIS-minus。软件开发环境：RedHat Linux 9.0, SQLite 3.3.4, StarSDB 1.0.2。

6.3 测试与比较

由于上述设计的综合查询算法既是 CPU 密集型的又是 I/O 密集型的，因此不能利用传统的关系数据库所采用的假设 I/O 代价远远大于 CPU 代价的方法，只能通过 I/O 代价来测量查询的平均响应时间，使用 CPU 代价和 I/O 代价共同来测量。

假设一个关系表中有 N 条记录，对其进行处理的 I/O 时间为 $T_{\mathrm{I/O}}$, CPU 时间为 T_{CPU}，则对每一条记录所进行的查询响应时间为

$$T_{\mathrm{average}}=(T_{\mathrm{I/O}}+T_{\mathrm{CPU}})/N$$

下面给出对上述的测试数据中点类型(POINT)数据的查询结果，各种查询算法平均查询响应时间如表 1 所示。

表 1 各种查询算法平均查询响应时间

记录个数/条	平均查询响应时间/ms		
	先空间串行查询算法	先属性串行查询算法	并行查询算法
12	2.845 2	2.432 5	1.530 7
84	18.982 1	17.658 4	11.142 9
90	19.031 5	18.365 4	11.482 7
151	32.002 4	30.257 4	20.547 2
947	176.325 6	169.257 3	112.628 6
1 004	184.296 3	178.125 8	119.012 8
1 196	200.658 4	198.321 9	132.160 4

根据表 1 可得出各种综合查询算法的性能比较曲线，如图 1 所示。

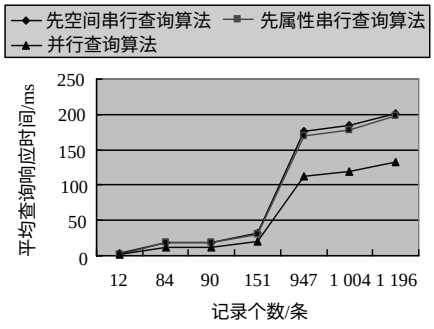

图 1 各种查询算法性能比较曲线

（下转第 64 页）